Multi-agent Pathfinding on Large Maps Using Graph Pruning: This Way or That Way? (Extended Abstract)*

Jiří Švancara,[1] Philipp Obermeier,[2,3] Matej Husár,[1] Roman Barták,[1] Torsten Schaub[2,3]

[1] Charles University, Prague, Czech Republic
[2] University of Potsdam, Potsdam, Germany
[3] Potassco Solutions, Potsdam, Germany
svancara@ktiml.mff.cuni.cz, phil@cs.uni-potsdam.de, husarmatej@gmail.com, bartak@ktiml.mff.cuni.cz, torsten@cs.uni-potsdam.de

Abstract

This paper extends a study on improving the performance of reduction-based solvers for the problem of multi-agent pathfinding. The task is to navigate a set of agents in a graph without collisions. Solvers that reduce this problem to other formalisms often have issues scaling to larger instances in terms of the graph size. A previous study suggests that pruning the graph of most vertices based on a randomly chosen shortest path for each agent. In this paper, we study the effect of different choices of these paths.

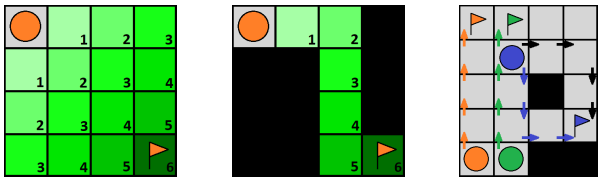
Introduction

The task of navigating a set of agents has numerous practical applications in robotics, logistics, automatic warehousing, and more. The theoretical abstraction of this task is Multi-agent Pathfinding (MAPF) where a set of agents moves discretely in a shared environment represented by a graph (Silver 2005). In this paper, we furthermore focus on finding a plan of the shortest length (i.e. minimizing the makespan, which is an NP-Hard problem (Yu and LaValle 2013)).

A common technique for solving MAPF optimally is to reduce it to another problem - most often SAT (Surynek 2016; Barták and Svancara 2019) or ASP (Erdem et al. 2013; Gebser et al. 2018). This type of solver suffers from instances where the underlying graph is large. The position of an agent is modeled by a variable representing a triple $(Agent, Position, Time)$, hence there is a blowup of the number of variables introduced to the solver. On the other hand, reduction-based solvers are successful on small and dense instances where the other popular approach - the search-based solvers (Sharon et al. 2012) - fails.

A previous study (Husár et al. 2022) suggested pruning large graphs of vertices that are likely unnecessary to solve the instance, thus lowering the number of variables created. The pruning is performed based on a randomly chosen shortest path for each agent, and only vertices near that path are taken into consideration. In this study, we extend that paper by exploring the effect of choosing multiple shortest paths for each agent and different ways to choose these paths.

*The original study was published at AAMAS 2022 (Husár et al. 2022).



(a) (b) (c)

Figure 1: (a) Motivation for pruning the graph. (b) Pruned graph. (c) Example where **C** finds suboptimal solution.

Graph Pruning for Reduction-based Solvers

In this paper, we describe the basic motivation and the strategies of graph pruning, but for the full report and proofs, we refer the reader to the original paper (Husár et al. 2022).

Motivation

A motivation for pruning graph can be seen in Figures (1a) and (1b) where an agent travels diagonally on a grid map. There are numerous paths the agent can take which may overwhelm the underlying solver. By pruning vertices outside of one random shortest path, the solver's search becomes much easier.

On the other hand, removing vertices may make the instance unsolvable if the vertices outside of the shortest paths are needed to ensure that the agents do not collide. For this reason, several strategies that iterate which vertices are pruned are proposed.

Pruning Strategies

The instance is created over a restricted graph G_k meaning that only vertices of distance at most k from a single random shortest path for each agent are used. To find the optimal makespan, we start with some lowerbound T and if there is no solution, we increase T by 1.

Baseline strategy (**B**) considers the whole graph and iteratively increases T. This is a commonly used approach that is complete and optimal.

Makespan-add strategy (**M**) considers G_1 and iteratively increases T. This strategy is neither complete nor optimal, as we may have deleted vertices necessary for a solution.

	type	B	P				C				M			
			SP	**AP**	**RP**	**DP**	**SP**	**AP**	**RP**	**DP**	**SP**	**AP**	**RP**	**DP**
Used vertices	*empty*	1,00	**0,14**	0,23	0,21	0,23	**0,15**	0,24	0,22	0,23	0,19	0,24	0,23	0,24
	maze	1,00	**0,18**	0,20	**0,18**	0,19	**0,20**	0,22	0,21	0,21	0,22	0,22	0,22	0,22
	random	1,00	**0,19**	0,27	0,24	0,25	**0,22**	0,30	0,28	0,29	0,25	0,31	0,30	0,31
	room	1,00	**0,21**	0,24	0,22	0,22	**0,23**	0,27	0,25	0,25	0,24	0,29	0,27	0,28
Solved instances	*empty*	0,78	**0,99**	0,81	0,84	0,82	**1,00**	0,81	0,84	0,82	0,87	0,81	0,82	0,80
	maze	0,85	0,87	**0,88**	0,87	0,87	**0,98**	0,97	0,97	**0,98**	0,94	0,94	0,94	0,94
	random	0,79	**0,91**	0,82	0,84	0,84	**1,00**	0,89	0,92	0,91	0,93	0,87	0,89	0,88
	room	0,80	**0,83**	0,81	0,82	0,82	**0,97**	0,92	0,95	0,94	0,89	0,89	0,89	0,89

Table 1: Ratio of used vertices, ratio of solved instances. The results are split by the map type.

Prune-and-cut strategy ($\mathbf{P}$) starts with $G_k, k = 0$ and iteratively increases k. If there is still no solution with the whole graph, T needs to be increased and $k = 0$ is set again. This strategy is both complete and optimal.

Combined strategy ($\mathbf{C}$) improves $\mathbf{P}$ by increasing both k and T at the same time, skipping many iterations. $\mathbf{C}$ is still complete but no longer optimal, as can be seen in Figure (1c), if the blue path is chosen at random.

Choosing Shortest Paths

The strategies may suffer from poor choices of the shortest paths (if there are multiple) as was demonstrated for strategy *combined*. To mitigate this, we introduce 4 approaches for choosing paths (union of vertices on those paths), over which G_k is built.

Single-path approach ($\mathbf{SP}$) is the one previously used. $\mathbf{SP}$ considers only a single random shortest path for each agent.

All-paths approach ($\mathbf{AP}$) tries to avoid example from Figure (1c) by considering all shortest paths for each agent. Indeed, such an example is no longer possible for the price of pruning fewer vertices. For example, no vertex is pruned in Figure (1a) by using $\mathbf{AP}$.

Random-paths approach ($\mathbf{RP}$) improves $\mathbf{AP}$ by considering only a random subset of the shortest paths to decrease the number of vertices while still giving the solver more options. The number of paths we consider is computed as $\frac{|\mathbf{AP}|}{|\mathbf{SP}|}$, meaning the ratio of vertices on the union of all shortest paths and the number of vertices on a single shortest path.

Distant-paths approach ($\mathbf{DP}$) further improves $\mathbf{RP}$ which may suffer from poorly choosing the subset of the shortest paths. Ideally, we prefer to choose paths that are diverse (not sharing many edges) and distant (the distance between vertices on the chosen paths is maximized). There is a polynomial algorithm that chooses diverse shortest paths but not distant (Hanaka et al. 2021). There is also an algorithm that finds near-optimal most distant paths (Häcker et al. 2021), but the problem is NP-Hard which is not desirable for a preprocessing function. Our proposed approach heuristically builds the paths iteratively. At each step, we try to add a new vertex to the currently built path and if there are multiple choices, we pick one that maximizes the minimal distance to all of the vertices currently on the found paths. Since this is just a heuristic, some examples make us choose an undesirable path because the approach greedily chooses the next vertex on the path without knowledge of the rest of the map. To mitigate this, we start to build the path from a different vertex rather than from the endpoints of the path.

Experiments

To test and compare the approaches to choosing the shortest paths, we performed a set of experiments on the commonly used benchmark set (Stern et al. 2019). The reduction-based solver used was an ASP encoding (Gebser et al. 2018) for which we used the grounding-and-solving system *clingo* (Kaminski et al. 2020) version 5.5.2. We ran the experiments on an Intel Xeon E5-2650v4 under Debian GNU/Linux 9, with each instance limited to 300s processing time and 28 GB of memory. The full implementation and results are available at `https://github.com/potassco/mapf-subgraph-system`.

Based on the results presented in Table 1, we observe that, unsurprisingly, using more than a single shortest path leads to an increase in the number of vertices in the restricted graph. This increase is more prominent in maps, where multiple shortest paths between two points exist. For example, for map type *maze* the difference is smaller.

The number of used vertices, thus the number of variables, is reflected in the runtime of the solver, and consequently in the number of solved instances within the timeout. The $\mathbf{SP}$ approach is the most successful in most cases, except for the *maze* maps, where the difference is much less prominent and for the $\mathbf{P}$ strategy $\mathbf{AP}$ outperforms $\mathbf{SP}$.

We conclude that 1) the initial motivation of pruning the graph as much as possible to improve the computation time is valid, and 2) considering more than a single shortest path usually does not lead to better performance.

On the other hand, we also observed that considering more paths leads to obtaining better plans for the sub-optimal strategies $\mathbf{C}$ and $\mathbf{M}$. The best approach in this aspect is naturally $\mathbf{AP}$ with $\mathbf{DP}$ and $\mathbf{RP}$ not far behind. Considering $\mathbf{SP}$ and $\mathbf{AP}$, the number of times $\mathbf{C}$ finds an optimal solution increases from 93% to 100%, 91% to 94%, 89% to 97%, and 86% to 93% for map types *empty*, *maze*, *random*, and *room*, respectively.

Acknowledgments

Research is supported by project P103-19-02183S of the Czech Science Foundation, the Czech-USA Cooperative Scientific Research Project LTAUSA19072, and DFG grant SCHA 550/15, Germany.

References

Barták, R.; and Svancara, J. 2019. On SAT-Based Approaches for Multi-Agent Path Finding with the Sum-of-Costs Objective. In Surynek, P.; and Yeoh, W., eds., *Proceedings of the Twelfth International Symposium on Combinatorial Search, SOCS 2019, Napa, California, 16-17 July 2019*, 10–17. AAAI Press.

Erdem, E.; Kisa, D. G.; Öztok, U.; and Schüller, P. 2013. A General Formal Framework for Pathfinding Problems with Multiple Agents. In desJardins, M.; and Littman, M. L., eds., *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence, July 14-18, 2013, Bellevue, Washington, USA*. AAAI Press.

Gebser, M.; Obermeier, P.; Otto, T.; Schaub, T.; Sabuncu, O.; Nguyen, V.; and Son, T. 2018. Experimenting with robotic intra-logistics domains. *Theory and Practice of Logic Programming*, 18(3-4): 502–519.

Häcker, C.; Bouros, P.; Chondrogiannis, T.; and Althaus, E. 2021. Most Diverse Near-Shortest Paths. In Meng, X.; Wang, F.; Lu, C.; Huang, Y.; Shekhar, S.; and Xie, X., eds., *SIGSPATIAL '21: 29th International Conference on Advances in Geographic Information Systems, Virtual Event / Beijing, China, November 2-5, 2021*, 229–239. ACM.

Hanaka, T.; Kobayashi, Y.; Kurita, K.; Lee, S. W.; and Otachi, Y. 2021. Computing Diverse Shortest Paths Efficiently: A Theoretical and Experimental Study. *CoRR*, abs/2112.05403.

Husár, M.; Svancara, J.; Obermeier, P.; Barták, R.; and Schaub, T. 2022. Reduction-based Solving of Multi-agent Pathfinding on Large Maps Using Graph Pruning. In Faliszewski, P.; Mascardi, V.; Pelachaud, C.; and Taylor, M. E., eds., *21st International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2022, Auckland, New Zealand, May 9-13, 2022*, 624–632. International Foundation for Autonomous Agents and Multiagent Systems (IFAAMAS).

Kaminski, R.; Romero, J.; Schaub, T.; and Wanko, P. 2020. How to build your own ASP-based system?! *CoRR*, abs/2008.06692.

Sharon, G.; Stern, R.; Felner, A.; and Sturtevant, N. R. 2012. Conflict-Based Search For Optimal Multi-Agent Path Finding. In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence, July 22-26, 2012, Toronto, Ontario, Canada.*

Silver, D. 2005. Cooperative Pathfinding. In *Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, 117–122.

Stern, R.; Sturtevant, N. R.; Felner, A.; Koenig, S.; Ma, H.; Walker, T. T.; Li, J.; Atzmon, D.; Cohen, L.; Kumar, T. K. S.; Barták, R.; and Boyarski, E. 2019. Multi-Agent Pathfinding: Definitions, Variants, and Benchmarks. In Surynek, P.; and Yeoh, W., eds., *Proceedings of the Twelfth International Symposium on Combinatorial Search, SOCS 2019, Napa, California, 16-17 July 2019*, 151–159. AAAI Press.

Surynek, P. 2016. Makespan Optimal Solving of Cooperative Path-Finding via Reductions to Propositional Satisfiability. *CoRR*, abs/1610.05452.

Yu, J.; and LaValle, S. M. 2013. Structure and Intractability of Optimal Multi-Robot Path Planning on Graphs. In *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence, July 14-18, 2013, Bellevue, Washington, USA*.